Relatoría HE2

Resumen:

Nuestro código trabaja con datos históricos desde 1960 hasta 2022 del PIB, recopilando información de todos los países año por año. Para cada país y año, el código determina si ese país pertenece al grupo de países ricos, clase media o pobres, utilizando una clasificación basada en percentiles de la riqueza mundial (dividida en tercios: 30%-30%-30%).
Con esta base de datos completa, dividimos los datos en dos conjuntos:

- 80% de los datos para entrenamiento (train), que abarca de 1960 hasta 2012.
- 20% de los datos para prueba (test), que va de 2013 hasta 2022.

El objetivo del programa es analizar la trayectoria de cada país dentro de estas categorías a lo largo del tiempo. Es decir, observa cuántos años un país ha sido clasificado como rico, de clase media o pobre, y con base en ese historial, intenta predecir su clasificación en 2022 dentro de las categorías actuales: High, Upper Middle, Middle, Lower Middle y Low Income.

Finalmente, evaluamos el desempeño del modelo utilizando métricas de validación sobre los datos de prueba, verificando qué tan bien puede predecir la clasificación de los países en 2022 basándose únicamente en su evolución histórica

Relatoría del Proyecto

1. Carga y Preparación de las Bases de Datos

1. Decidimos comenzar con la carga y arreglo de las bases de datos. David fue el principal responsable de este proceso.
2. No teníamos experiencia trabajando con Kaggle, por lo que tuvimos que aprender nuevas técnicas (anteriormente lo habíamos hecho con GitHub).
3. Luego de subir los datos, enfrentamos el reto de unir ambas bases en un solo diccionario para proceder con la limpieza y posterior análisis.

2. Limpieza y Distribución del Trabajo

4. Una vez completada la limpieza de las bases de datos, discutimos cómo dividir el trabajo para avanzar de manera eficiente. Durante esta etapa, notamos que no había valores nulos en las bases de datos.
5. Se acordó la siguiente distribución de tareas:
    - **David**: Responsable de la base de datos.
    - **Lucas y Louis**: Construcción de redes neuronales.
    - **Juan Fernando y Alejandro**: Redacción del *Readme*, relatoría y desarrollo de los bonos.

6. En el cuadernillo, Lucas, Louis y David documentaron los problemas encontrados en la elaboración del código.

3. Reunión de Avance y Corrección de Errores

7. Se agendó una reunión para el jueves a las 8 p.m. con el objetivo de finalizar el trabajo.
8. Antes de la reunión, identificamos errores en la limpieza de los datos y los corregimos para poder construir las redes neuronales.
9. Encontrar una base de datos adecuada para el bono fue un reto, hasta que hallamos una con datos del PIB per cápita. Luego, trabajamos en integrarla correctamente con nuestra base de datos existente.
10. Durante la reunión, Louis presentó el trabajo de limpieza de datos realizado junto a Lucas, explicando el proceso y los desafíos encontrados.

4. Problemas Técnicos y Soluciones en Redes Neuronales

11. Se presentaron las principales dificultades:

- La categorización por año y percentiles tomó mucho tiempo.
- La implementación de *One-Hot Encoding* requería ajustes para funcionar correctamente. Al empezar a construir las redes neuronales, nos dimos cuenta de que los valores booleanos (*True/False*) no se adaptaban a los *inputs* requeridos en el modelo, por lo que tuvimos que modificar estos valores a una matriz con valores binarios (*1/0*), permitiendo que los países cambiaran de categoría más fácilmente en el tiempo.
- La codificación de las variables *X* e *Y* fue compleja, ya que los *strings* debían transformarse en valores numéricos.

12. Probamos el código y verificamos que el punto 1 funcionaba correctamente, por lo que avanzamos con el punto 2 y los bonos, distribuyéndolos de la siguiente manera:

- **David**: Bono del punto 3.
- **Louis y Lucas**: Bonos del punto 2.
- **Juan Fernando y Alejandro**: Bono del anexo.

13. Cerramos la reunión y cada integrante se enfocó en su tarea asignada.
14. Grabamos la reunión para futuras consultas y la compartimos en el grupo de WhatsApp.

5. Problemas y Mejoras Implementadas

15. Juan Fernando tuvo varios inconvenientes con el bono y tardó 4 horas en completarlo.
16. Al día siguiente, Louis informó dificultades con la pregunta 2.b, pero logró resolverla tras varias horas de trabajo.
17. Tuvimos un gran problema al correr el modelo en la GPU.

2.a Segunda Corrección del Modelo de Redes Neuronales

Problemas Identificados

- **Baja Precisión Inicial:**
  El modelo base tenía un accuracy de aproximadamente 47%, con métricas de precisión, recall y f1-score bajas en algunas clases, lo que indicaba que la arquitectura inicial no capturaba la complejidad de los datos.
- **Espacio de Hiperparámetros Limitado:**
  Los valores iniciales para la arquitectura (número de capas y neuronas), la regularización (`alpha`), la tasa de aprendizaje y el número de iteraciones eran insuficientes para encontrar una configuración óptima.
- **Métricas de Evaluación Inadecuadas:**
  Se utilizaba `accuracy` como métrica, lo que no resultó ideal en presencia de clases desbalanceadas.

Soluciones Aplicadas

- **Ampliación del Espacio de Búsqueda de Hiperparámetros:**
  - Se evaluaron nuevas configuraciones de `hidden_layer_sizes`, incluyendo opciones con dos y tres capas (por ejemplo, `(100, 100)`, `(150, 100)` y `(100, 50, 50)`).
  - Se amplió el rango para `alpha` desde 1e-4 hasta 1.
  - Se expandieron los valores de `learning_rate_init` (desde 0.0001 hasta 0.1).
  - Se incluyó `max_iter` con valores de 500, 1000 y 1500.
  - Se añadió la opción de `learning_rate` con los valores `'constant'` y `'adaptive'`.
- **Implementación de Early Stopping:**
  Se activó `early_stopping=True` en el modelo para evitar el sobreentrenamiento.
- **Validación Cruzada Mejorada y Cambio de Métrica:**
  Se utilizó 5-fold cross-validation junto con la métrica `f1_macro` en GridSearchCV.

Impacto y Beneficios

- **Incremento del Accuracy:**
  Se aumentó el accuracy de aproximadamente 47% a 79%.
- **Optimización de la Convergencia:**
  Se encontró una configuración óptima sin sobreentrenarse.

Problemas y Soluciones en el Punto 2.b

19. El modelo presentaba sobreajuste, evidenciado en los siguientes síntomas:

- Alta precisión en entrenamiento, pero fluctuaciones en validación.
- Pérdida en validación errática, con valores extremadamente altos.
- Problemas numéricos en la pérdida, indicando desajuste en el *learning rate*.

- Sesgo del modelo hacia las clases mayoritarias.

20. Para solucionar estos problemas, realizamos varias mejoras:

- Verificamos la normalización de los datos.
- Reducimos la tasa de aprendizaje inicial.
- Aumentamos el *dropout* para disminuir el sobreajuste.
- Implementamos *L2 regularization* y *EarlyStopping*.

21. Como resultado, se obtuvieron mejoras significativas en el modelo:

- La pérdida inicial pasó de 168.8 a 34.4 y finalizó en 1.68.
- La pérdida en validación mejoró de 6.45 a 2.03, finalizando en 1.03.
- La precisión inicial subió de 26.9% a 30.3%.
- La precisión en validación mejoró del 37.17% en versiones previas hasta un 51.86% en la versión final.

22. Ajustes realizados por versiones:

- **Primera versión**: Modelo inicial sin ajustes.
- **Segunda versión**: Reducción de *learning rate*, agregado *BatchNormalization* y aumento de *dropout*.
- **Versión final**: Uso de *StandardScaler*, ajuste de inicialización de pesos, reducción de *dropout* y aplicación de *ReduceLROnPlateau*.

Problemas y Soluciones en el Punto 2.c

23. La primera versión tenía una configuración incorrecta:

- Se usó MSE en lugar de *categorical_crossentropy*, lo que afectó la clasificación.
- La precisión inicial se mantenía en 25%, con pérdida inicial de 0.3743 y final de 0.3705, lo que indicaba que la red no estaba aprendiendo.

24. **Primera corrección**:

- Reducción del *learning rate* a 0.01.
- Implementación de *BatchNormalization*.
- Agregado de capas adicionales.
- La precisión aumentó al 72.84%, con una pérdida de 0.0883, pero la función de pérdida aún no era la adecuada.

25. **Segunda corrección**:

- Cambio a *categorical_crossentropy*.
- Ajuste del *learning rate* a 0.001.
- Ajuste del *dropout* y uso de *EarlyStopping*.

- La precisión final en validación mejoró a 69.84% y la pérdida final a 0.6478.

2.d Visualización de SHAP (Beeswarm Plot)

Problemas Encontrados

1. **Error de Indexación (`TypeError`)**
2. **`numpy.ndarray` Object has no attribute 'columns'`**
3. **NameError en la Variable del Modelo**

Soluciones Aplicadas

1. **Definir los Nombres de las Columnas**
2. **Convertir Arrays NumPy a DataFrames**
3. **Corregir el Nombre del Modelo**
4. **Pasar el DataFrame y/o feature_names en `summary_plot`**

6. Desarrollo de los Bonos y Resultados

6.1 Primer Bono: Integración de una Nueva Base de Datos

18. Al trabajar en el bono, inicialmente no lo comprendimos correctamente y asumimos que se trataba de todo el anexo. Tras discutirlo con Camilo y el equipo, aclaramos que solo era necesario incluir otra base de datos.
19. Convertimos las variables numéricas a categóricas para entrenar el modelo y clasificar los países en categorías.

6.2 Segundo Bono: Análisis con SHAP y Waterfall Plot

20. Ejecutamos un código que genera un explainer de SHAP y un *waterfall plot* para visualizar cómo cada variable influye en la predicción del modelo.
21. **Resultados**:

- Se observó que las variables con mayores valores absolutos influyen más en la predicción.

7. Conclusión

- El modelo inicial no aprendía debido a una mala configuración.
- La optimización de hiperparámetros permitió alcanzar un *accuracy* del 70% en validación.
- La inclusión de una nueva base de datos para el bono permitió mejorar la clasificación de países en categorías.
- El uso de SHAP permitió entender mejor la contribución de cada variable en el modelo.